IoT Product Design and Rapid Prototyping

Part B

Brian Rashap

January 2023

Copyright ©2022 CNM.

Before We Get Started

- Create an account at `openweathermap.org`
- Generate an API key at: `https://home.openweathermap.org/api_keys`

Copyright ©2022 CNM.

Clear Out Credentials

Going forward we will need to connect to the Internet and the Particle Cloud:

- Run L09_00_HelloReset
  - Clear out all credentials (to remove IoTClassroom)
  - Reconnect to DDCIOT
- Add your home credentials, either by:
  - Add your home credentials to L09_00_HelloReset, or
  - When you get home, use *particle serial wifi* to reconnect to your home wifi
- Code also will show you your stored credentials, your IP and MAC address, and the Access Point into.

Module 9 - The Cloud

The Cloud

Cloud Computing
Servers
Virtual Desktop
Software Platform
Applications
Storage / Data
OS
Apps
Router
Switch
End User

MQTT: MQ Telemetry Transport

Adafruit.io

Let's create an Adafruit.io account.

MQTT Elements explained

- TheClient - object that defines the TCP (Transmission Control Protocol) connection over WiFi.
- mqtt - object that defines the MQTT connection using the WiFi object, the MQTT server/port, and user name/password.
- FeedName - a "variable" located on Adafruit.io that can be subscribed or published to. There can be many of these.
- mqttObj - object that will be used in the C++ code that will be used to publish or subscribe to an Adafruit.io feed. There needs to be one object for each feed.
- value - Variable in the C++ code that stores information to be published or to receive information from a feed that is subscribed to.

NOTE: FeedName, mqttObj, and value should be given descriptive "names" similar to the naming convention for all variables and objects in the C++ code.

MQTT Elements in VSCode

```
#include <Adafruit_MQTT.h>
#include "Adafruit_MQTT/Adafruit_MQTT_SPARK.h"
#include "Adafruit_MQTT/Adafruit_MQTT.h"

#include "credentials.h"
/*  Copy the Adafruit.io Setup line and hte next four lines to a credentials.h file
//************************* Adafruit.io Setup ****************************************
#define AIO_SERVER      "io.adafruit.com"
#define AIO_SERVERPORT  1883        // use 1883 for SSL
#define AIO_USERNAME    "username"  // replace with your Adafruit.io username
#define AIO_KEY         "key"       // replace with your Adafruit.io key
*/

/************ Global State (you don't need to change this!) ***   ***************/
TCPClient TheClient;

// Setup the MQTT client class by passing in the WiFi client and MQTT server and login
     details.
Adafruit_MQTT_SPARK mqtt(&TheClient,AIO_SERVER,AIO_SERVERPORT,AIO_USERNAME,AIO_KEY);

/****************************** Feeds ***************************************/
// Setup Feeds to publish or subscribe
// Notice MQTT paths for AIO follow the form: <username>/feeds/<feedname>
Adafruit_MQTT_Subscribe subFeed = Adafruit_MQTT_Subscribe(&mqtt,AIO_USERNAME "/feeds/
     feed1");
Adafruit_MQTT_Publish pubFeed = Adafruit_MQTT_Publish(&mqtt,AIO_USERNAME "/feeds/feed2");

/************Declare Variables*************/
unsigned int last, lastTime;
float subValue,pubValue;

```

MQTT Publish and Subscribe

```
void setup() {
  Serial.begin(9600);
  waitFor(Serial.isConnected, 15000); //wait for Serial Monitor to startup

  WiFi.connect();                     //Connect to internet, but not Particle Cloud
  while(WiFi.connecting()) {
    Serial.printf(".");
  }

  mqtt.subscribe(&subFeed);          // Setup MQTT subscription for subFeed feed.
}

void loop() {
  // Publishing to a MQTT feed
  if(mqtt.Update()) {   //if mqtt object (Adafruit.io) is available to receive data
    Serial.printf("Publishing %0.2f to Adafruit.io feed FeedNameB \n",value1);
    pubfeed.publish(value1);
  }
    // Two new functions that will be useful:
    // atof() - ASCII to Float: converts an ASCII string to a floating point number
    // atoi() - ASCII to Integer: converts an ASCII string to an integer

  // Receive data from a subscription to an MQTT feed
  Adafruit_MQTT_Subscribe *subscription;
  while ((subscription = mqtt.readSubscription(100))) { //wait a moment for new feed data
    if (subscription == &subFeed) {  // assign new data to appropriate variable
      value2 = atof((char *)subFeed.lastread); //value2 = data from MQTT subscription
      Serial.printf("Received %0.2f from Adafruit.io feed FeedNameB \n",value2);
    }
  }
}
```

Assignment: L09_01_SubscribePublish

1. Modify the starter code for your Adafruit.io
2. Publish
   - Publish a random number to a feed once every 6 seconds (do not use a delay).
   - Create a line chart on your dashboard to display the random number.
3. Subscribe
   - Add a button to your Adafruit.io dashboard and connect it to a feed called buttonOnOff.
   - Subscribe to the buttonOnOff and turn on the on board LED (D7) when pressed.
4. Experiment with other blocks
   - Replace the button with a slider.
   - Control the brightness of an LED.
   - Display data with other dashboard blocks.

Local and Static Variables

```
const int TEMPFREQ = 10000, MOISTFREQ = 30000, MOISTPIN = A3;
float tempC;
int moist;
void loop() {
  tempC = getTemp(TEMPFREQ);
  moist = getMoisture(MOISTPIN,MOISTFREQ);
}

float getTemp(int timeInterval) {
  int currentTime;
  static int lastTime = -999999;
  static float data;

  currentTime = millis();
  if(currentTime - lastTime > timeInterval) {
    lastTime = millis();
    data = BME.readTemperature();
  }
  return data;
}

int getMoisture(int probePIN, int timeInterval) {
  static int lastTime = -999999;
  static int data;

  if(millis() - lastTime > timeInterval) {
    lastTime = millis();
    data = analogRead(probePIN);
  }
  return data;
}
```

JSON

JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. It is based on a subset of the JavaScript Programming Language Standard ECMA-262 3rd Edition - December 1999. JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others. These properties make JSON an ideal data-interchange language.
JSON is built on two structures:

- A collection of name/value pairs. In various languages, this is realized as an object, record, struct, dictionary, hash table, keyed list, or associative array.
- An ordered list of values. In most languages, this is realized as an array, vector, list, or sequence.

JSON Parser Generator

Creating objects in JSON are straightforward but can be tedious. There is a JSON Generator available to simplify the process.

```
#include <JsonParserGeneratorRK.h>

void createEventPayLoad(float tempValue, float presValue, float humValue) {
  JsonWriterStatic<256> jw;
  {
    JsonWriterAutoObject obj(&jw);

    jw.insertKeyValue("Temperature", tempValue);
    jw.insertKeyValue("Pressure", presValue);
    jw.insertKeyValue("Humidity", humValue);
  }
  Particle.publish("env-vals",jw.getBuffer(), PRIVATE);
}
```

Assignment: L09_02_GPSPublish

1. Every 10 seconds (without using delays), generate random GPS coordinates within Albuquerque.
2. Publish to Adafruit.io using MQTT and JSON format. Use "lat" and "lon" as the names for the JSON data elements.
3. Using the Map block to create a dashboard that shows the GPS coordinates

Webhooks

WEBHOOKS vs APIs
P
R
N
D
S
Webhooks are automatic
APIs are manual
API
User sends and requests data
Webhook
Data sent automatically
User Interface
Server

Step 1 - OpenWeather

- Create an account at `openweathermap.org`
- Generate an API key at: `https://home.openweathermap.org/api_keys`

You can generate as many API keys as needed for your subscription. We accumulate the total load from all of them.

Key	Name	Status	Actions
07f56344e3fe7a27974a52eb54783b71	Default	Active	
dacc7f9f41f4cca0a274cf925b97a356	IoTClass	Active	

Create key

API key name

Generate

Step 2 - Create Webhook

From `console.particle.io`:

Step 3 - Select Custom Template

Step 4 - Update Custom Template with API format

Adapted from `https://openweathermap.org/api/one-call-api` and `https://openweathermap.org/current`

```
{
    "event": "GetWeatherData",
    "responseTopic": "{{PARTICLE_DEVICE_ID}}/{{PARTICLE_EVENT_NAME}}",
    "url": "https://api.openweathermap.org/data/2.5/onecall",
    "requestType": "GET",
    "noDefaults": true,
    "rejectUnauthorized": true,
    "responseTemplate": "{\"lat\":{{lat}},\"lon\":{{lon}},\"dt\":{{current.dt}},\"temp
     \":{{current.temp}},\"uvi\":{{current.uvi}},\"clouds\":{{current.clouds}},\"ws\":{{
     current.wind_speed}},\"wd\":{{current.wind_deg}} }",
    "query": {
        "lat": "{{lat}}",
        "lon": "{{lon}}",
        "exclude": "minutely,hourly,daily,alerts",
        "units": "metric",
        "appid": "dacc7f9f41f4cca0a274cf925b97a356"
    }
}
```

Step 5 - Particle Argon Code

```
const char *EVENT_NAME = "GetWeatherData";
unsigned int lastTime;
const float lat=35.0045, lon=-106.6465;  //update to your favorite location

void setup() {
    Serial.begin(9600);
    waitFor(Serial.isConnected,15000);
    String subscriptionName = String::format("%s/%s/", System.deviceID().c_str(), EVENT_NAME);
    Particle.subscribe(subscriptionName, subscriptionHandler, MY_DEVICES);
    Serial.printf("Subscribing to %s\n", subscriptionName.c_str());
}

void loop() {
    if((millis() - lastTime) > 60000) {
        Serial.printf("\n\nTime = %i\n",millis());
        Particle.publish(EVENT_NAME, "", PRIVATE);
        Particle.publish(EVENT_NAME, String::format("{\"lat\":%0.5f,\"lon\":%0.5f}", lat, lon), PRIVATE);
        lastTime = millis();
    }
}

void subscriptionHandler(const char *event, const char *data) {
    JSONValue outerObj = JSONValue::parseCopy(data);
    JSONObjectIterator iter(outerObj);
    while(iter.next()) {
            Serial.printf("key=%s value=%s\n",(const char *) iter.name(),(const char *) iter.value().toString());
    }
}
```

Assignment: L09_03_GetWeather

1. Using the OpenWestherMap webhook get the outside weather conditions.
2. Display the OLED:
   - GPS location (hard coded)
   - Indoor conditions from BME280
   - Current outdoor conditions.

Module 10 - Semiconductors

Semiconductors

Semiconductor

- A silicon atom has four electrons in its outer shell and bonds tightly with four surrounding silicon atoms creating a crystal matrix with eight electrons in the outer shells. The tight bonds make pure silicon non-conducting.
- Phosphorus has five electrons, and when combined, the fifth electron becomes a "free" electron that moves easily within the crystal when a voltage is applied.
- Boron has only three electrons in its outer shell and can bond with only three of surrounding silicon atoms. Thus one silicon atom has a vacant location in its outer shell, called a "hole," that readily accepts an electron.

pn junction diode

- When p-type and n-type semiconductors are bonded, holes and free electrons are attracted, combine, and disappear near the boundary. Since there are no carriers in this area, it is called a depletion layer and it is an insulator.
- A positive voltage applied to the p-type region causes electrons to flow sequentially from the n-type. The electrons will first disappear by combining with holes, but excess electrons will move to the positive pole and current will flow.

Bipolar Junction Transistor

The transistor has three regions, namely base, emitter and collector. The emitter is a heavily doped terminal and emits electrons into the base. The base terminal is lightly doped and passes the emitter-injected electrons on to the collector. The collector terminal is intermediately doped and collects electrons from base. This collector is large as compared with other two regions so it dissipates more heat.

Bipolar Junction Transistor - Modes of Operation

- **Saturation:** Current freely flows from collector to emitter. (ON Switch)
- **Cut-off:** No current flows from collector to emitter. (OFF Switch)
- **Active:** The current from collector to emitter is proportional to the current flowing into the base. (Amplifier)
- **Reverse-Active:** Like active mode, the current is proportional to the base current, but it flows in reverse from emitter to collector (not the purpose transistors were designed for).

NPN Modes

Voltage relations	NPN Mode	PNP Mode
$V_E < V_B < V_C$	Active	Reverse
$V_E < V_B > V_C$	Saturation	Cutoff
$V_E > V_B < V_C$	Cutoff	Saturation
$V_E > V_B > V_C$	Reverse	Active

Water Analogy

Active Mode NPN Transistor Circuit

If you apply a voltage V_{IN} that is high enough to forward-bias the base-to-emitter junction, current (I_B) will flow from the input terminal, through R_B, through the BE junction, to ground. Current (I_C) will also flow through R_C and the collector-to-emitter portion of the transistor.

NOTE: V_{OUT} is an amplified but inverted signal of V_{IN}.
This simple circuit will step-up a 0 - 3.3V output from the microcontroller to 0 - 5.0V (inverted). The low impedance of the output will also provide sufficient current to drive a higher current device (e.g., a relay).

PNP Transistor

NPN Transistors are more common than PNP for a number of reasons:

- The voltage and current behavior of an NPN transistor is significantly more intuitive.
- When a switch or driver circuit is required, NPNs provide a more straightforward interface to digital output signals (such as a control signal generated by a microcontroller).
- NPNs are higher performance (faster switching speeds) due to higher mobility of electrons vs holes.

Logic Voltage Level Standards

Data source: EETimes, A brief recap of popular logic standards (Mark Pearson, Maxim).

Or, what is wrong with the NeoPixels.

Emitter Follower - Saturation and Cutoff Mode

- NeoPixels are designed around 5V CMOS transistors.
  - $V_{IH} > 3.5V$
  - 3.3V Microcontroller $V_{OH} = 3.3V$
- An Emitter Follower (i.e., a PNP transistor wired backwards) is a current amplifier, but will also produce a $V_{OUT} = 3.9V$.
- Alternatively, the first NeoPixel could be sacrificed by reducing its V_{cc} to 4.3V with a diode.

Emitter Follower Layout (using 2N3906 PNP Transistor)

Fritzing

Schematic

Typical NPN Pre-Amplifier Circuit

This is referred to as a Common Emitter amplifier as the Emitter ground is common to both the input and output voltage. The Common Emitter amplifies both voltage and current.

Two Stage Pre-Amplifier

Ohm's Law - Revisited

PreAmp vs PowerAmp

PreAmp:

- A preamp boosts the signal up to 'line level'.
- Guitar PreAmp
  - A pure guitar signal typically sounds weak and anaemic, as is seen if a guitar is directly plugged PA system.
  - A preamp is able to raise a guitar's signal up to an audible volume.
  - It can also be used to affect the audio characteristics.

PowerAmp:

- A power amp boosts that line level signal even more – so that it can be projected through speakers.

Power Amplifier

Op Amp Lesson

Inverting Op Amp

$$A = \frac{V_{out}}{V_{in}} = -\frac{R_f}{R_{in}}$$

Non-inverting Op Amp

$$A = \frac{V_{out}}{V_{in}} = 1 + \frac{R_f}{R_{in}}$$

Power the OpAmp with $V^+ = 12V$ and $V^- = 0$

Assignment: L10_Semiconductor

1. L10_01_NeoPixel
   - Add a Emitter-Follower into your NeoPixel circuit to boost the pixel commands to 5V.
2. L10_02_NPNAmp
   - Using the DAC and code from L05_00_lowPass to create a sine wave output (reduce the amplitude from 127.5 to 30).
   - Amplify using an NPN preamp.
   - Measure the circuit at each node using the oscilloscope.
3. L10_03_OpAmp
   - Replace the NPN preamp with a non-inverting LM741 Op Amp. Use a potentiometer for R_{in}.

L10_04_MysteryCircuit (Extra Credit)

- Layout circuit in Fritzing - there is no Argon in this circuit
- Build and test circuit with input at Oscilliscope station
- Record output voltage for all combinations of buttons presses
- Figure out what it is doing and why it works.

Power the OpAmp with $V^+ = 12V$ and $V^- = -12V$

L10_05_MysteryCircuit2 (Extra Credit)

- Set analogWrite frequency to 50k
- AnalogWrite 1.8V and see the result of different button presses.
- Try different analogWrite voltages

Power the OpAmp with $V^+ = 12V$ and $V^- = 0V$

Module 11 - Sensors

Piezoelectric Elements

- The piezoelectric effect is the appearance of electrical potential (voltage) across the side of a crystal when subject to mechanical stress.
- Conversely, a crystal becomes mechanically stressed (deformed in shape) when a voltage is applied across opposite faces.
- By utilizing an analogRead(), the voltage (and thus the amount of pressure on the crystal) can be measured.

Assignment: Carnival Game

- Notebook: schematic
- Fritzing diagram
- Wire your circuit
- Write the code

1. L11_01_HighStriker
   - Connect the piezo sensor and the NeoPixel tower to the Argon
   - Each time the piezo is struck, find the maximum voltage generated.
   - Light up the NeoPixel tower proportional to the force the piezo is struck with.

Load Cells

1. A load cell is a force transducer. It converts a force such as tension, compression, pressure, or torque into an electrical signal that can be measured and standardized. As the force applied to the load cell increases, the electrical signal changes proportionally. The most common types of load cells used are hydraulic, pneumatic, and strain gauge.
2. The HX711 module is a precision 24-bit analog-to-digital converter (ADC) designed for weigh scales and industrial control applications to interface directly with a bridge sensor.

Wheatstone Bridge

1. The Wheatstone bridge was invented by Samuel Hunter Christie in 1833 and improved and popularized by Sir Charles Wheatstone in 1843.
2. A Wheatstone bridge is an electrical circuit used to measure an unknown electrical resistance by balancing two legs of a bridge circuit, one leg of which includes the unknown component.
3. The primary benefit of the circuit is its ability to provide extremely accurate measurements (in contrast with something like a simple voltage divider).

HX711A Library

```
// From the Command Palette install the HX711A library, that will give you HX711.h
#include "HX711.h"
HX711 myScale(DT,CLK);     // any two digital pins

const int CAL_FACTOR=1000; //changing value changes get_units units (lb, g, ton, etc.)
const int SAMPLES=10; //number of data points averaged when using get_units or get_value

float weight, rawData, calibration;
int offset;

void setup() {
  myScale.set_scale();           // initialize loadcell
  delay(5000);                   // let the loadcell settle
  myScale.tare();                // set the tare weight (or zero)
  myScale.set_scale(CAL_FACTOR); //adjust when calibrating scale to desired units
}

void loop() {
  // Using data from loadcell
  weight = myScale.get_units(SAMPLES); // return weight in units set by set_scale();
  delay(5000)                          // add a short wait between readings

  // Other useful HX711 methods
  rawData = myScale.get_value(SAMPLES);// returns raw loadcell reading minus offset
  offset = myScale.get_offset();       // returns the offset set by tare();
  calibration = myScale.get_scale();   // returns the cal_factor used by set_scale();
}
```

Assignment: L11_Sensor (Learn to Calibrate)

- Notebook: schematic
- Fritzing diagram
- Wire your circuit
- Write the code

1. L11_02_Scale
   - Set initial CAL_FACTOR to 1000 and measure a known weight. (Note: one cup of water (in a paper cup) is approx. 244 g).
   - Adjust CAL_FACTOR until you get the expected measurement in grams.
   - Post data to Adafruit.io and/or ThingSpeak™
   - Optional: Send text via IFTTT.

More DataTypes - Strings, strings, and char[]

```
// A string (lowercase 's') is an array of characters
  char lastName[7] = "Rashap";
  char firstName[6] = {'B','R','I','A','N'};
  char name[12];

  //The "*" indicates a pointer, which we will learn about later
  char *myName = "Brian";

// A String is a Class that holds a character array
  String instructor = "BRIAN RASHAP";

void setup() {
  Serial.begin();

  Serial.printf("lastName = %s, %i\n",lastName,sizeof(lastName));
  Serial.printf("firstName = %s, %i\n",firstName,sizeof(firstName));
  Serial.printf("name = %s, %i\n",name,sizeof(name));
  Serial.printf("myName = %s, %i\n",myName,sizeof(myName));

  // We can not use %s for the variable instructor, why?
}
```

```
Serial monitor opened successfully:
lastName = Rashap, 7
firstName = BRIAN, 6
name = , 12
myName = Brian, 4
```

Too Much Time On My Hands

When the Particle Argon connects to the Particle Cloud, it synchronizes its clock to the current time.

```
// Declare Global Variables in Header
String DateTime, TimeOnly;

void setup() {
  Time.zone(-7);            // MST = -7, MDT = -6
  Particle.syncTime();      // Sync time with Particle Cloud
}

void loop() {
  DateTime = Time.timeStr();            //Current Date and Time from Particle Time class
  TimeOnly = DateTime.substring(11,19); //Extract the Time from the DateTime String

  //%s prints an array of char
  //the .c_str() method converts a String to an array of char
  Serial.printf("Date and time is %s\n",DateTime.c_str());
  Serial.printf("Time is %s\n",TimeOnly.c_str());

  delay(10000); //only loop every 10 seconds
}
```

To learn more about the String Class: `https://www.arduino.cc/reference/en/language/variables/data-types/stringobject/`

Soil Moisture Sensors

Resistive Sensor

Capacitive Sensor

Assignment: Moisture Probe

- Notebook: schematic
- Fritzing diagram
- Wire your circuit
- Write the code

1. L11_03_Moisture
   - Using the Capacitive Soil Moisture probe, in your notebook note the moisture readings when:
     - Empty Cup
     - Submerged in water to the notch
     - Dry Soil
     - Soil after watered
   - Display the moisture to the OLED with a Time-stamp.

Seeed Sensors

Seeed Grove - Dust Sensor

Seeed Grove - Air Quality Sensor v1.3

Shinyei PPD42NS low-cost dust sensor

MP-503 Air Quality Sensor

Seeed Assignment

- Notebook: schematic
- Fritzing diagram
- Wire your circuit
- Write the code

1. L11_04_SeeedSensors
   - Look up the Seeed sensors online to see how they work.
   - Do not blindly copy the examples. Only use the code you need.
   - By looking at the .cpp code, determine how to get a quantitative value for air quality, in addition to the qualitative level.
   - Display air quality and particulate concentration to an Adafruit.io dashboard.

Midterm 2 - House Plant Watering System

Inductors

- Current flowing in a wire produces a magnetic field (B) around the wire (from Ampere's Law).
- Wire wrapped into a coil produces a magnetic field that resembles a bar magnet through the center of the coil.
- Also, in a coil, this magnetic field produces an effect known as Inductance (L) that opposes changes in electric current.

Relays

- When a device (e.g. a pump) requires higher voltage ($> 5V$) or higher current, then a relay can be used as a switch for the device
- The relay is activated by a digital pin from the microcontroller.
  - However, as the relay requires 100mA from the digital pin. To provide sufficient current, use a current amplifying emitter follower to draw current directly from the USB connection (V_{BUS}).

Optocoupled Relay

- Optocoupler isolates the relay load (which could be up to 240V) from the microcontroller electronics.

Smart Houseplant Watering System

2N3906 Emitter Follower

Design

1. Components:
   - 2N3906 Emitter Follower and Relay
   - BME280 and SEEED sensors
   - OLED Display
2. Publish soil moisture and room environmental data to a new dashboard.
3. Automatically water your plant when the soil is too dry.
   - Only turn on the pump for a very short period of time ($\frac{1}{2} sec$).
4. Integrate a button into your dashboard that manually waters the plant.

Relay and Pump Fritzing Diagram

If V_{BUS} doesn't provide enough power, add external supply.

Midterm 2 Continued

1. Integrate the entire system
   - Create a user friendly Adafruit.io dashboard
   - Buy and/or create a structure to hold the plant, pump, and sensors.
   - Integration of SMS/email messaging using Zapier (following slides).
   - Video demo your Plant Watering System
2. Add to your portfolio
   - Add the project to your Hackster.io feed.
   - Create your own Github repository with a copy of your final project folder.
   - Don't forget the .gitignore
3. Class presentation - hackster, video demo, dashboard

Online Integrations Using Zapier

Zapier is an online automation tool that connects your favorite apps, such as Outlook, Slack, Mailchimp, and more. You can connect two or more apps to automate repetitive tasks.

Zapier Example

```
Adafruit_MQTT_Subscribe *subscription;
while ((subscription = mqtt.readSubscription(10000))) {
  if (subscription == &gotmail) {
    Serial.printf("You Got Mail \n");
    digitalWrite(D7, HIGH);
    flagServo.write(90);
    delay(10000);
    digitalWrite(D7, LOW);
    flagServo.write(5);
  }
}
```

https://zapier.com/apps/adafruit-io/integrations/sms

1. Create an Zapier account using your cnm.edu email and sign up for the 14-day "Professional" trial.
2. Use the Zapier instructions in class_materials to light up your D7 LED when new mail arrives in your cnm.edu Outlook account.
3. Add to your Midterm project to send yourself an SMS text whenever your soil is too dry. In Zapier:
   - Create Feed Trigger from Adafruit.io.
   - Add a "Only Continue If...".
   - Add a Send to SMS action.

Module 12 - Memory: Bit, Bytes, and More

Counting Sheep

Negative Numbers

Question: How are negative numbers represented in binary?

Type	Storage size	Value range
char	1 byte	-128 to 127 or 0 to 255
unsigned char	1 byte	0 to 255
signed char	1 byte	-128 to 127
int	2 or 4 bytes	-32,768 to 32,767 or -2,147,483,648 to 2,147,483,647
unsigned int	2 or 4 bytes	0 to 65,535 or 0 to 4,294,967,295
short	2 bytes	-32,768 to 32,767
unsigned short	2 bytes	0 to 65,535
long	4 bytes	-2,147,483,648 to 2,147,483,647
unsigned long	4 bytes	0 to 4,294,967,295

Answer: Left-most bit is 1 to signify negative. But wait...

2's Compliment

2's compliment is used as it makes the math consistent.

Integer		2's Complement
Signed	Unsigned	
5	5	0000 0101
4	4	0000 0100
3	3	0000 0011
2	2	0000 0010
1	1	0000 0001
0	0	0000 0000
-1	255	1111 1111
-2	254	1111 1110
-3	253	1111 1101
-4	252	1111 1100
-5	251	1111 1011

The negative plus the positive equals zero.

Bitwise Operations

The following table lists the Bitwise operators supported by C. Assume variable 'A' holds 60 and variable 'B' holds 13, then –

Operator	Description	Example
&	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) = 12, i.e., 0000 1100
|	Binary OR Operator copies a bit if it exists in either operand.	(A | B) = 61, i.e., 0011 1101
^	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) = 49, i.e., 0011 0001
~	Binary One's Complement Operator is unary and has the effect of 'flipping' bits.	(~A) = ~(60), i.e,. 1100 0011
<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 = 240 i.e., 1111 0000
>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 = 15 i.e., 0000 1111

Bit Shifting

Logical Bit Shift

Arithmetic Bit Shift

Whether the logical or arithmetic right shift is used depends on the datatype of the variable (unsigned or signed).

BONUS: But what about Floating Point

IEEE-754 Floating Point

Example: In scientific notation: $-36382.36 = -3.638236 * 10^4$
With binary exponential equals $-1 * 1.1103014945983887 * 2^{15}$

- Sign: Negative = 1
- Exponent: 15 = 10001110 (with an exponent bias of 10000000)
- Mantissa: 1110301494598387 = 00001110000111100101110 0

Floating point representation is: 11000111000011100001111001011100

Electrically Erasable Programmable Read Only Memory

EEPROM emulation allows small amounts of data to be stored and persisted even across reset, power down, and user and system firmware flash operations. Since the data is spread across a large number of flash sectors, flash erase-write cycle limits should not be an issue in general.

```
len = EEPROM.length(); //available EEPROM bytes
// Argons have 4096 bytes of emulated EEPROM.
// Addresses 0x0000 through 0x0FFF

addr = 0x00AE;    //addr between 0 and len-1

val = 0x45;
EEPROM.write(addr, val);

value = EEPROM.read(addr);
```

Pointers

1. A pointer is a variable whose value is the address of another variable.
2. When you declare a pointer, the $*$ symbol denotes that this variable is a pointer variable. For example:
   - Pointer to an Integer: int *ptr;
3. Reference operator (&) gives the address of a variable.
4. To get the value stored in the memory address, we use the dereference operator ($*$).

Pointers

```
int data = 13;
int data2;
int *ptr;

void setup() {
  Serial.begin(9600);
  delay(1000);
  ptr = &data;          //point ptr to the memory location of data
  data2 = *ptr;         //set data2 to value of data (13)

  // Print the Value and Address of the Variables
  Serial.printf("Variable         Value      Address \n");
  Serial.printf("  data            %i       0x%X   \n",data, &data);
  Serial.printf("  ptr           0x%X       0x%X   \n",ptr, &ptr);
  Serial.printf("  data2           %i       0x%X   \n",data2,&data2);
}
```

```
Serial monitor opened successfully:
Variable          Value       Address
  data              13        0x2003E380
  ptr          0x2003E380     0x2003E3F4
  data2             13        0x2003E3F0
```

Pointers and Arrays

```
int arr[] = {100, 200, 300};

void loop() {
  // Compiler converts below to *(arr + 2).
  Serial.printf("%i \n", arr[2]);

  // So below also works.
  Serial.printf("%i \n", *(arr + 2));
}
```

When an array (arr[]) is declared, the variable is a pointer to the first element of a continuous block of memory. In this case there are 3 elements, each 4-bytes in size, for a total of 12-bytes.

Finding Average of an Array

```
// This function finds the average of an array.
// The array is passed to it as a pointer.

float getAverage(int *array,int size) {
  int j;
  float total=0;
  for(j=0;j<size;j++) {
    total += array[j];
  }
  return total/size;
}
```

Finding Average of Arrays in Action

```
int xArray[4], yArray[256];
int *pointerX, *pointerY;
float average;
int i, sizeX, sizeY;

void setup() {
  pointerX=&xArray[0];
  sizeX=sizeof(xArray)/4;
  pointerY=&yArray[0];
  sizeY=sizeof(yArray)/4;
  for(i=0;i<sizeX;i++) {
    xArray[i] = random(0,255);
  }
  for(i=0;i<sizeY;i++) {
    yArray[i] = random(256,512);
  }
  average = getAverage(pointerX, sizeX);
  average = getAverage(pointerY, sizeY);
}
```

```
Array X Average = 162.50
Array Y Average = 388.35
xArray[0] value: 173, *pointerX:     173, pointerX:   0x2003E3DC
xArray[1] value: 179, *(pointerX+1): 179, pointerX+1: 0x2003E3E0
xArray[2] value: 110, *(pointerX+2): 110, pointerX+2: 0x2003E3E4
xArray[3] value: 188, *(pointerX+3): 188, pointerX+3: 0x2003E3E8
```

Returning Multiple Values from a Function

Arguments can be passed to a function by reference; thus, allowing multiple parameters to be returned by a function.

```
const int BILLBUTTON = D4;
float table1, table2;  // subtotal for each table
float tax1, tax2, tip1, tip2;
bool billReady;

void calcBill(float subtotal, float *tax, float *suggestedTip);

void setup() {
  // NOT SHOWN: Setup of Serial Monitor, pinMode, etc.
  table1 = 23.76;
  table2 = 47.29;
}

void loop() {
  billReady = digitalRead(BILLBUTTON);
  if(billReady) {
    calcBill(table1,&tax1,&tip1);// pass by reference, memory address sent to function
    calcBill(table2,&tax2,&tip2);// the function uses pointers to place values in memory
    Serial.printf("Table1\nSubtotal:%0.2f\nTax:%0.2f\nSuggested Tip:%0.2f\n",table1,tax1,
     tip1);
  }
}

void calcBill(float subtotal, float *tax, float *suggestedTip) {
  *tax = subtotal * 0.07;
  *suggestedTip = subtotal * 0.18;
}
```

memcpy(), (char *), and strtol()

```
int color;
byte data[] = {0x23,0x42,0x41,0x34,0x32,0x35,0x44,0x39,0x35};
byte buf[6];


/* memcpy() - copy from specific memory locations to new locations
 * memcpy(to, from, size);
 *     to -> pointer to starting address of where to copy to
 *     from -> pointer to starting address of where to copy from
 *     size -> number of btyes to copy
 */

memcpy(buf, &data[1],6);     //copy bytes 1 through 6 and place in buf

/* (char *) - typecasting a data type to a char-type pointer */

Serial.printf("Converting the data array to ascii symbols returns %s,\n",(char *)data);


/* strtol() - string to long - similar to atoi()
 * strtol(charString, end, base)
 *     charString -> string that contains number to be converted
 *     end -> character to end conversion on (set to NULL)
 *     base -> base of integer (16 for hex)
 */

color = strtol((char *)buf,NULL,16); // convert string to int (hex)
```

EXAMPLE: Adafruit MQTT Subscribe - Color Picker

```
int color;
byte buf[6];

  Adafruit_MQTT_Subscribe *subscription;
  while ((subscription = mqtt.readSubscription(1000))) {
    if (subscription == &mqttColor) {
      Serial.printf("Received from Adafruit: %s \n",(char *)mqttColor.lastread);
      memcpy(buf, &mqttColor.lastread[1],6);      //strip off the '#'
      Serial.printf("Buffer: %s \n",(char *)buf);
      color = strtol((char *)buf,NULL,16);        // convert string to int (hex)
      Serial.printf("Buffer: 0x%02X \n",color);
    }
  }
```

L12_Memory Assignments

- Notebook: schematic
- Fritzing diagram
- Wire your circuit
- Write the code

1. L12_01_ColorPicker
   - Create a feed and dashboard on Adafruit.io using the Color Picker block.
   - Subscribe to your Color Picker feed and convert the lastRead() to a integer (hex)
   - Light up your NeoPixel ring the received color.
   - Using pointers create a function to convert the hex color into individual R,G,B components using Bit Shifting and AND.
   - From void loop, store the components of the color as bytes in the Argon's EEPROM.
2. L12_02_RetrieveShow
   - Retrieve the color from EEPROM memory.
   - Convert to a hex color code (for example 0xABCDEF)
   - Display the color on the NeoPixel ring using setPixelColor(n,hexColor)

Useful Properties: Identity Element

An identity element is a special type of element of a set with respect to a binary operation on that set, which leaves any element of the set unchanged when combined with it.

Addition:

$$x + 0 = x \tag{1}$$

Multiplication:

$$x * 1 = x \tag{2}$$

Bitwise AND:

$$x \& 1 = x \tag{3}$$

Bitwise OR:

$$x | 0 = x \tag{4}$$

Added Bonus: Array of Functions via Pointers

```
//Array of pointers to each of the functions
int (* funky[4])(int x, int y) = {add,sub,mult,divi};

int a,b,answer,i;

void setup() {
  Serial.begin(9600);
}

void loop() {
  a = random(0,100);
  b = random(0,100);
  for(i=0;i<4;i++) {
    answer = funky[i](a,b);
    Serial.printf("For function %i: a = %i and b = %i equals %i \n",i,a,b,answer);
    delay(250);
  }
  Serial.printf("\n\n\n");
  delay(3000);
}

// The Functions
int add(int x,int y) {return x+y;}

int sub(int x,int y) {return x-y;}

int mult(int x,int y) (return x*y;}

int divi(int x,int y) {return x/y;}
```

Module 13 - Motion

MPU6050 Accelerometer

- Data Output - 16 Bit
- Gyros range: $\pm$ 250 500 1000 2000 °/s
- Accel range: $\pm$ 2 $\pm$ 4 $\pm$ 8 $\pm$ 16g

- The interrupt pin notifies the MPU about available data. To reduce power consumption, the processor can go into sleep mode and the interrupt can be used to wake up the processor.
- XDA and XCL refer to the I2C bus that the MPU-6050 controls, so it can read from slave devices such as magnetometers etc.

Accelerometers

DIP Switches and Register Maps

Remember: serial usb setup-done

Table 18: Memory map

Register Name	Address	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0	Reset state
hum_lsb	0xFE	hum_lsb<7:0>								0x00
hum_msb	0xFD	hum_msb<7:0>								0x80
temp_xlsb	0xFC	temp_xlsb<7:4>				0	0	0	0	0x00
temp_lsb	0xFB	temp_lsb<7:0>								0x00
temp_msb	0xFA	temp_msb<7:0>								0x80
press_xlsb	0xF9	press_xlsb<7:4>				0	0	0	0	0x00
press_lsb	0xF8	press_lsb<7:0>								0x00
press_msb	0xF7	press_msb<7:0>								0x80
config	0xF5	t_sb[2:0]			filter[2:0]				spi3w_en[0]	0x00
ctrl_meas	0xF4	osrs_t[2:0]			osrs_p[2:0]			mode[1:0]		0x00
status	0xF3					measuring[0]			im_update[0]	0x00
ctrl_hum	0xF2						osrs_h[2:0]			0x00
calib26..calib41	0xE1...0xF0	calibration data								individual
reset	0xE0	reset[7:0]								0x00
id	0xD0	chip_id[7:0]								0x60
calib00..calib25	0x88...0xA1	calibration data								individual

Registers:	Reserved registers	Calibration data	Control registers	Data registers	Status registers	Chip ID	Reset
Type:	do not change	read only	read / write	read only	read only	read only	write only

REMINDER - Data Types: Numbers

Data Type	8-bit AVR systems (Arduino Uno)			32-bit ARM systems (Teensy 3.2)		
	bytes	range (signed)	range (unsigned)	bytes	range (signed)	range (unsigned)
char	1	-128 to 127	0 to 255	1	-128 to 127	0 to 255
short	2	+/- 32,767	0 to 65,353	2	+/- 32,767	0 to 65,353
int	2	+/- 32,767	0 to 65,353	4	+/- 2,147,483,648	0 - 4,294,967,295
long	4	+/- 2,147,483,648	0 - 4,294,967,295	4	+/- 2,147,483,648	0 - 4,294,967,295
long long	8	+/- 9,223,372,036,854,770,000	0 to 18,446,744,073,709,551,615	8	+/- 9,223,372,036,854,770,000	0 to 18,446,744,073,709,551,615
float	4	3.4E +/- 38 (7 digits)	n/a	4	3.4E +/- 38 (7 digits)	n/a
double	4	3.4E +/- 38 (7 digits)	n/a	8	1.7E +/- 308 (15 digits)	n/a
long double	8	1.7E +/- 308 (15 digits)	n/a	8	1.7E +/- 308 (15 digits)	n/a
Unambiguous						
uint8_t	1	n/a	0 to 255	1	n/a	0 to 255
int8_t	1	-128 to 127	n/a	1	-128 to 127	n/a
uint16_t	2	n/a	0 to 65,353	2	n/a	0 to 65,353
int16_t	2	+/- 32,767	n/a	2	+/- 32,767	n/a
uint32_t	4	n/a	0 - 4,294,967,295	4	n/a	0 - 4,294,967,295
int32_t	4	+/- 2,147,483,648	n/a	4	+/- 2,147,483,648	n/a

There are $7.5x10^{18}$ grains of sand on Earth. A long long integer and floating point numbers are larger than this.

Initializing MPU6050

```
// Initialize the MPU in the void setup()
void setup() {
  // Begin I2C communications
  Wire.begin();

  // Begin transmission to MPU-6050
  Wire.beginTransmission(MPU_ADDR);

  // Select and write to PWR_MGMT1 register
  Wire.write(0x6B);
  Wire.write(0x00); // wakes up MPU-6050

  // End transmission and close connection
  Wire.endTransmission(true);
}
```

Reading Acceleration Data from the MPU-6050

```
// Declare variables
byte accel_x_h, accel_x_l;    //variables to store the individual btyes
int16_t accel_x;              //variable to store the x-acceleration

// Set the "pointer" to the 0x3B memory location of the MPU and wait for data
Wire.beginTransmission(MPU_ADDR);
Wire.write(0x3B); // starting with register 0x3B
Wire.endTransmission(false); // keep active.

// Request and then read 2 bytes
// Syntax:
//    Wire.requestFrom(I2C_addr, quantity, stop);
//    Wire.read(); //repeat this for each byte to be read

Wire.requestFrom(MPU_ADDR, 2, true);
accel_x_h = Wire.read(); // x accel MSB
accel_x_l = Wire.read(); // x accel LSB

accel_x = accel_x_h << 8 | accel_x_l;     // what happens if declared int instead?
Serial.printf("X-axis acceleration is %i \n",accel_x);
```

Note: the data is stored in Big Endian Byte Order. The most significant byte (the "big end") of the data is placed at the byte with the lowest address. The rest of the data is placed in the next byte.

Assignment: L13_Motion

- Notebook: schematic
- Fritzing diagram
- Wire your circuit
- Write the code

1. L13_01_MPU6050
   - Read values from the memory addresses associated with X, Y, and Z acceleration.
   - Convert the returned acceleration values to standard gravity units (e.g. when flat on the table, a_z = -1G).
2. L13_02_AutoRotate
   - Display date and time on an OLED display.
   - Use accel values to auto-rotate the OLED.
3. Extra
   - Modify L13_01_MPU6050 to be able to modify range/sensitivity with a button or encoder.

Pythagorean Theorem

The Pythagorean Theorem

$$c^2 = a^2 + b^2$$

$$a = \sqrt{c^2 - b^2}$$

$$b = \sqrt{c^2 - a^2}$$

$$c = \sqrt{a^2 + b^2}$$

SOH CAH TOA

- sin = opposite over hypotenuse
- cos = adjacent over hypotenuse
- tan = opposite over adjacent

cos(Θ) = adjacent / hypotenuse
or
Adjacent = hypotenuse * cos(Θ)

sin(Θ) = opposite / hypotenuse
or
opposite = hypotenuse * sin(Θ)

Gravity and Orientation

Assignment: L13_Motion

- Notebook: schematic
- Fritzing diagram
- Wire your circuit
- Write the code

1. L13_03_Airplane
   - Calculate pitch $\theta = -asin(a_x)$.
   - Calculate roll $\phi = atan2(a_y, a_z)$.
2. L13_04_Shock
   - Store a_{tot} in an array every 10ms for 5s.
   - Find and print the max value from the array.
   - Repeat.
   - Note: When at rest, the $a_{tot} \approx 1$.

Recall, that trigonometric functions return radians which needs to be converted to degrees. See the Unit Circle slide in L02_HelloLED.

Pitch and Roll Improved

$$\text{Pitch}(\Theta) = \arctan\left(\frac{a_x}{\sqrt{{a_y}^2+{a_z}^2}}\right)$$

$$\text{Roll } (\Phi) = \arctan\left(\frac{a_y}{\sqrt{{a_x}^2+{a_z}^2}}\right)$$

$$\text{Yaw } (\Psi) = \arctan\left(\frac{\sqrt{{a_x}^2+{a_y}^2}}{{a_z}^2}\right)$$

Assignment: L13_Motion (Extra Credit)

1. Modify L13_03_Airplane
   - Improve L13_03_Airplane with the more detailed trigonometric equations.
   - Use the NeoPixel ring to visualize roll.
   - When holding the accelerometer a fixed distance ($\approx 6in$) from the NeoPixel tower, change pitch and light up the pixel it is pointing at.
   - Note: for the 3rd bullet use TOA from SOH-CAH-TOA

Stepper Motors

28BYJ Stepper Motor

- 2048 steps per revolution
  - 32 steps per rotor revolution
  - Gear ratio 1:64
- Capable of 10-15 RPM (at 5V)
- ULN2003 Darlington Array driver

Stepper.h

- Stepper myStepper (spr,IN1,IN3,IN2,IN4)
- myStepper.setSpeed(speed)
- myStepper.step(steps)

ULN2003 Darlington Array

A Darlington Array is a set of current amplifying circuits that take outputs from the microcontroller and boost the current used to drive the motor.

Gyroscopes

The gyroscope measures angular velocity (degrees per second).

Acceleration, Velocity, and Position

The Reimann Sum method can be used to "integrate" acceleration to velocity and velocity to position.

- Gyroscope output is angular velocity: ω $(\frac{degrees}{second})$
- To get change in angular position ($\Delta\theta$), multiple each angular velocity by the time step: $\Delta\theta = \omega * \Delta t$
- Use the Reimann Sum to get the resulting angular position

$$\theta = \sum_{t=0}^{t=T} (\omega * \Delta t)$$

Deadband

A deadband is a band of input values that in a control system create an output that is zero.

Assignment: L13_Motion

- Notebook: schematic
- Fritzing diagram
- Wire your circuit
- Write the code

1. L13_05_DuckGoRound
   - Place your duck onto the stepper motor.
   - Wire the stepper motor noting the order: IN1, IN3, IN2, IN4.
   - In void loop(), move the motor 2 rotations clockwise, pause, 1 rotation counter-clockwise, repeat.
2. L13_06_DriveByWire
   - Connect the MPU-6050 to your system.
   - Obtain the z-axis rotation from the appropriate register on the MPU-6050. Convert to angular rotation (°per sec).
   - Calculate the angular position (°) of the gyroscope.
   - Have the stepper motor track movement in the gyroscope.

Hall Effect Sensor

Discovered by Edwin Hall in 1879, the Hall Effect is the production of a voltage difference (the Hall voltage) across an electrical conductor, transverse to an electric current in the conductor and to an applied magnetic field perpendicular to the current.

Interrupts

Interrupts are a way to write code that is run when an external event occurs. As a general rule, interrupt code should be very fast, and non-blocking. This means performing transfers, such as I2C, Serial, TCP should not be done as part of the interrupt handler. Rather, the interrupt handler can set a variable which instructs the main loop that the event has occurred.

```
pinMode(pin, INPUT); \\ can also use INPUT_PULLUP or INPUT_PULLDOWN
attachInterrupt(pin, function, mode);
```

Mode: defines when the interrupt should be triggered. Three constants are predefined as valid values:

- CHANGE to trigger the interrupt whenever the pin changes value,
- RISING to trigger when the pin value goes from low to high,
- FALLING for when the pin value goes from high to low.

Software Timers

There are also software timer interrupts. The Argon can manage up to 10 timers simultaneously.

```
Timer timer(1000,printEverySecond);

void setup() {
  Serial.begin(9600);
  timer.start();
}

void printEverySecond() {
    static int count = 0;
    Serial.printf("count=%i, time = %u ms \n", count,millis());
    count++;
}
```

Note:

- The timer callback is similar to an interrupt - it shouldn't block.
- Multiple timers are serviced sequentially when several timers trigger simultaneously, thus requiring special consideration when writing callback functions.

Assignment: L13_Motion

- Notebook: schematic
- Fritzing diagram
- Wire your circuit
- Write the code

1. L13_07_Alarm
   - Connect Hall Effect Sensor, button, and Neopixel to simulate an alarm system.
   - Use the button to enable / disable alarm.
   - When armed:
     - Neopixel is GREEN when magnet detected.
     - Blinking RED when magnet not detected.
2. L13_08_RPM
   - Place magnet on shaft of lathe.
   - Create an interrupt function that returns the time per rotation using the Hall Effect Sensor.
   - Convert this time to rotations per minute.
   - Display on Adafruit.io databoard.
   - EXTRA:
     - Create a speedometer using a servo motor. (The Servo class natively available).
     - Measure RPM on other equipment at FUSE.

Module 14 - DataXfer

Serial Peripheral Interface

- Master Out, Slave In (MOSI) connects to Data In
- Master In, Slave Out (MISO) connects to Data Out

Multiple SPI Modules

- SPI uses the $\overline{CS}$ lines to select which peripheral is active.
- Having two SPI devices selected at the same time causes interference.
- In void setup(), always initalize all SPI devices as "off"
  - Note: $\overline{CS}$ is active LOW ("off" is HIGH)

μSD Card Module

1. SD cards are sensitive to interface to the pins
   - Keep wires short
   - Data lines need to be 3.3V, use level shifter is needed
2. FAT16 or FAT32 format
   - File naming - 8.3 (e.g., myfile12.csv)
3. Pinout
   - ◀ 5V - Power input(3.3V or 5V)
   - 3V output to power other devices
   - GND - Ground
   - ◀ CLK - Clock
   - ▶ DO - MISO
   - ◀ DI - MOSI
   - ◀ CS - Chip Select
   - CD - Card Detect

RC Time Constant

- Up until now, we have considered wires as ideal conductors; however:
  - Wires have non-zero resistance, longer and thinner wires have more resistance.
  - When two wires are close to each other, there is a parasitic capacitance between them.
- The time constant ($\tau = \frac{1}{RC}$) of an RC circuit determines the amount of time it takes a square wave input to reach 63% of it's final value.
- Some communication protocols expect sharp transitions between 0 and 3.3V for clock and data signals.

Piezoelectric Elements - Revisited

- The piezoelectric effect is the appearance of electrical potential (voltage) across the side of a crystal when subject to mechanical stress.
- Conversely, a crystal becomes mechanically stressed (deformed in shape) when a voltage is applied across opposite faces.
- By utilizing an analogRead(), the vibration (change in mechanical stress) can be monitored over time.

Structural Engineering Sensors

FAT File System - SDCard Argon Project

Feature	FAT32	NTFS
Maximum Partition Size	2TB	2TB
Maximum File Size	4GB	16TB
Maximum File Name	8.3 Characters	255 Characters
File/Folder Encryption	No	Yes
Fault Tolerance	No	Auto Repair
Security	Network Only	Local and Network
Compression	No	Yes
Compatibility	Win 95/98/2000/XP and the derivations	Win NT/2000/XP/Vista/7 and the later versions

The FAT (File Allocation Table) file system, originally designed in 1977 for floppy disks, is simple and robust. It offers good performance in very light-weight implementations, but does not deliver performance, reliability and scalability afforded by modern file systems (such as NTFS or exFAT).

Note: FAT file name follows a 8.3 format (e.g., ABCDEFGH.txt). We will be appending two digits, so our base name can be up to 6 characters (e.g. FILE_BASE_NAME = "mydata" → mydata42.csv).

Galaxy S9+ Vibration Analysis

Using the FFT Tutorial in Class_Materials repository

	A	B	C	D	E
1	TimeStamp	Signal	Frequency	FFT Magnitude	Complex FFT
2	0.00061	2.62	0.4209321	2.000009766	4096.02
3	0.00119	1.93	0.8418642	0.836477508	-552.77950380473+1621.47055713326i
4	0.00177	0.96	1.2627963	0.298364661	-414.982558995766-448.522671528173i
5	0.00235	0.13	1.6837284	0.270681692	353.443826952842-427.069259698417i
6	0.00293	0	2.1046606	0.083873335	-72.9068609990069+155.532672030055i
7	0.003509	0	2.5255927	0.129856545	252.456289478309+83.6253876318606i
8	0.004089	0	2.9465248	0.255636434	-516.88842919695+83.210943362584i
9	0.004669	0	3.3674569	0.360085774	423.28074046682-603.882664071708i
10	0.005249	0.14	3.788389	0.379410535	88.80273300538+771.941714466294i
11	0.005829	1.26	4.2093211	0.04066392	-41.3210095359081-72.3054897410451i
12	0.006409	2.16	4.6302532	0.051383144	94.6397062012862-46.0135075977235i
13	0.006988	2.57	5.0511853	0.084507321	25.9352596801745-171.116717569232i
14	0.007568	1.89	5.4721175	0.041174283	36.0724144460193-76.2199128809511i
15	0.008148	0.73	5.8930496	0.04976769	-73.9953870941929-70.0944447984849i

The Galaxy S9 vibrates at 159.11 Hz

Assignment: L14_DataXfer

- Notebook: schematic
- Fritzing diagram
- Wire your circuit
- Write the code

1. L14_01_Vibration
   - Connect the piezo sensor, a button, and a μSD module to your Argon.
   - Each time button is pressed, execute a loop 4096 times:
     - Every 500μsec, collect piezoelectric data (without using a delay).
     - Save the piezo data and a timestamp (converting micros() to seconds) to a 2-dimensional array.
   - When the loop is complete, write the timestamp and data to a file.
   - Collect vibration data from the lathe, cell phone vibration, other machines at FUSE.
   - Use Excel and the FFT Tutorial (class_slides) to resample graph data in frequency domain (This process will be reviewed as a class.).

Assignment: L14_DataXfer EXTRA

- Notebook: schematic
- Fritzing diagram
- Wire your circuit
- Write the code

1. L14_01_Vibration
   - Borrow a microphone.
   - Install in place of the piezo sensor.
   - Use Physics Toolbox Sensor Suite - Tone Generator to create a tone of a specific frequency.
   - Record/save with the Argon, and create an FFT

Bluetooth

- The Bluetooth protocol operates at 2.4GHz in the same unlicensed ISM frequency band where RF protocols like ZigBee and WiFi also exist.
- Bluetooth networks (commonly referred to as piconets) use a master/slave model to control when and where devices can send data. In this model, a single master device can be connected to up to seven different slave devices. Any slave device in the piconet can only be connected to a single master.

Bluetooth - Generic Access Profile

- The Generic Access Profile (GAP) controls connections and advertising in Bluetooth. GAP is what makes your device visible to the outside world, and determines how two devices can (or can't) interact with each other.
- GAP defines various roles for devices, but the two key concepts to keep in mind are Central devices and Peripheral devices.
  - Peripheral devices are small, low power, resource constrained devices that can connect to a much more powerful central device. Peripheral devices are things like a heart rate monitor, a BLE enabled proximity tag, etc.
  - Central devices are usually the mobile phone or tablet that you connect to with far more processing power and memory.

Bluetooth - Generic Attribute Profile (GATT)

BLE GATT Client-Server model

- Generic Attribute Profile defines the way that two BLE devices transfer data back and forth using concepts called Services and Characteristics. It makes use of a generic data protocol called the Attribute Protocol (ATT), which is used to store Services, Characteristics and related data in a simple lookup table using 16-bit IDs for each entry in the table.
- GATT comes into play once a dedicated connection is established between two devices, meaning that you have already gone through the advertising process.

Bluetooth - Services and Profiles

- A Profile is a pre-defined collection of Services. The Heart Rate Profile, for example, combines the Heart Rate Service and the Device Information Service.
- Services break data up into logic entities, and contain specific chunks of data called characteristics. A service can have one or more characteristics, and each service distinguishes itself from other services with a unique numeric ID called a UUID, which can be either 16-bit (official BLE Services) or 128-bit (custom services).
- A Characteristic contains a single data point or an array of related data. For example: X/Y/Z values of an accelerometer.

ASCII Reminder

- ASCII characters (the symbols that we are use to reading) can be represented by a single byte (uint8_t).

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

ASCII: American Standard Code For Information Interchange

Argon BLE - UART Service

```
// These UUIDs were defined by Nordic Semiconductor and are now the defacto standard for
// UART-like services over BLE. Many apps support the UUIDs now, like the Adafruit Bluefruit app.
const BleUuid serviceUuid("6E400001-B5A3-F393-E0A9-E50E24DCCA9E");
const BleUuid rxUuid("6E400002-B5A3-F393-E0A9-E50E24DCCA9E");
const BleUuid txUuid("6E400003-B5A3-F393-E0A9-E50E24DCCA9E");

BleCharacteristic txCharacteristic("tx", BleCharacteristicProperty::NOTIFY, txUuid, serviceUuid);
BleCharacteristic rxCharacteristic("rx", BleCharacteristicProperty::WRITE_WO_RSP, rxUuid, serviceUuid, onDataReceived, NULL);
BleAdvertisingData data;

//onDataReceived is used to receive data from Bluefruit Connect App
void onDataReceived(const uint8_t* data, size_t len, const BlePeerDevice& peer, void* context) {
  uint8_t i;

  Serial.printf("Received data from: %02X:%02X:%02X:%02X:%02X:%02X \n", peer.address()[0], peer.address()[1],peer.address()[2], peer.address()[3], peer.address()[4], peer.address()[5]);
  Serial.printf("Bytes: ");
  for (i = 0; i < len; i++) {
    Serial.printf("%02X ",data[i]);
  }
  Serial.printf("\n");
  Serial.printf("Message: %s\n",(char *)data);
}
```

Argon BLE - UART Transmit Example

```
const int UART_TX_BUF_SIZE = 20;
uint8_t txBuf[UART_TX_BUF_SIZE];
uint8_t i;

SYSTEM_MODE(SEMI_AUTOMATIC); //Using BLE and not Wifi

void setup() {
    Serial.begin();
    waitFor(Serial.isConnected, 15000);

    BLE.on();
    BLE.addCharacteristic(txCharacteristic);
    BLE.addCharacteristic(rxCharacteristic);
    data.appendServiceUUID(serviceUuid);
    BLE.advertise(&data);

    Serial.printf("Argon BLE Address: %s\n", BLE.address().toString().c_str());
}

void loop() {
    for(i=0;i<UART_TX_BUF_SIZE-1;i++) {
      txBuf[i] = random(0x40,0x5B); //Captial ASCII characters plus @
    }
    txBuf[UART_TX_BUF_SIZE-1] = 0x0A;
    txCharacteristic.setValue(txBuf, UART_TX_BUF_SIZE);
    for(i=0;i<UART_TX_BUF_SIZE;i++) {
        Serial.printf("%c",txBuf[i]);
    }
    delay(5000);
}
```

Formatted Print to a Buffer

```
// sprintf does a formatted print to a buffer of type char[

const int BUFSIZE = 50;
byte buf[BUFSIZE];
int people;
float dogs,avg;

void setup() {
  Serial.begin(9600);
  people = 5;
  dogs = 13;
}

void loop() {
  avg = dogs / people;
  sprintf((char *)buf,"The %i people have on average %0.2f dogs \n",people, avg);
  Serial.printf("Buf contains the string: %s",(char *)buf);
}
```

The buffer can then be used to as the data payload between devices, for example, using BlueTooth.

Note: Windows treats a *n* as a LF-CR (0x0D0A), if a LF is needed (e.g., for BLE) then it needs to be inserted manually:

```
buf[BUFSIZE-1] = 0x0A;
```

Assignment: L14_BlueTooth

- Notebook: schematic
- Fritzing diagram
- Wire your circuit
- Write the code

L14_03_BlueTooth

- Load Bluefruit Connect on your smart device. Using the code on the proceeding slides, establish BLE UART communications
- Attach the encoder and NeoPixel tower to your Argon.
- Repeat the NeoPixel ring/strip assignment (L04_02_NeoPixel) on your Argon.
- When it changes, send the the encoder or NeoPixel position via BLE to Bluefruit Connect.
- Reset the encoder and NeoPixel ring to the appropriate state when values 0-29 are received from Bluefruit Connect.

Assignment: L14_BlueTooth - Colors

ColorPicker String
5 bytes plus CR

['!'] ['C'] [byte red] [byte green] [byte blue] [CRC]

L14_03_BlueTooth (Continued)

- Plotter function in Bluefruit Connect:
  1. Every time the encoder moves, generate and change the pixels to a random color (R,G,B format, not Hex).
  2. Plot the pixel number and three RGB components on the Bluefruit Plotter.
- Controller -> Color Picker screen on Bluefruit Connect:
  1. Send a color to the Argon.
  2. Identify in your code if ColorPicker string or general UART string is received.
  3. If ColorPicker, then using bitwise left shift and OR to convert string to hex color similar to L12_02_RetrieveShow
  4. Change your Neopixel color to match Color Picker

Assignment: L14_BlueTooth - Colors (EXTRA)

L14_03_BlueTooth (EXTRA)

- Experiment sending signals from the Bluefruit Connect Control Pad
- Use the Up/Down arrows to change the neopixel brightness
- Use the Left/Right arrows to cycle through a rainbow
- Code in a different neopixel effect for each of the number keys

Global Positioning System

- 29 satellites (24 active plus 5 reserve) at an altitude of 12550 miles, circling the earth twice per day.
- Envisioned by Aerospace Corporation 1963
- First satellite 1973
- Open to commercial use 1985
- GPS receiver measures time it takes signal (at speed of light) to get from satellite to receiver.
- Uses triangulation from at least 4 satellites to obtain latitude, longitude, altitude, and time.
- GNSS is an international system of satellites that includes GPS and others

Global Positioning System

- Miniature GPS module
- Houses a complete GPS/GNSS solution
- Both I2C and UART interfaces
- STEMMA interface for easy prototyping

Adafruit_GPS Library

```
$GPGGA,202410.000,4042.6000,N,07400.4858,W,1,4,3.14,276.7,M,-34.2,M,,*63
$GPRMC,202410.000,A,4042.6000,N,07400.4858,W,0.08,161.23,160412,,,A*70
$GPGGA,202411.000,4042.5999,N,07400.4854,W,1,3,17.31,275.8,M,-34.2,M,,*5D
$GPRMC,202411.000,A,4042.5999,N,07400.4854,W,0.14,161.23,160412,,,A*7A

Time: 20:24:11.0
Date: 16/4/2012
Fix: 1 quality: 1
Location: 4042.5998N, 7400.4853W
Speed (knots): 0.14
Angle: 161.23
Altitude: 275.80
Satellites: 3
```

- Call gps.read() at the beginning of void loop()
- Then immediately call GPS.parse(GPS.lastNMEA()) to make the data available
- When you need it, you can than access GPS.latitude, GPS.longitude, GPS.speed, etc.

LoRa

LoRa is a long range, low power, inexpensive technology for Internet of Things

LoRa Range vs Data Rate

LoRa uses license-free sub-gigahertz radio frequency ISM bands in the deployed region such as 868 MHz in Europe and 915MHz in North America.

LoRa Features

LoRa was many desirable features:

- It has very wide coverage range about 5 km in urban areas and 15 km in suburban areas
- Battery lifetime up to 15 years
- One LoRa gateway takes care of thousands of nodes.
- Easy to deploy and low cost.
- Enhanced the secure data transmission by embedded end-to-end AES128 encryption

RXLR896 LoRa Module

Features:

- Semtech SX1276 Engine
- Excellent blocking immunity
- Low receive current
- High sensitivity
- Control easily by AT commands
- 127 dB Dynamic Range RSSI
- Designed with integrated antenna
- AES128 Data encryption

AT Commands

AT commands are commands which are used to control the modems where AT stands for Attention. These commands were derived from Hayes commands which were used by the Hayes smart modems. Every wireless modem requires an AT command to interact with a computer machine.

Communication between the Argon and RYLR896 takes place over UART (Serial1) using the same "Serial" commands that were used in Lesson 3 and Lesson 4.

Batteries - Going Mobile

1. All Particle platforms have JST-PH pins for a Lithium Polymer (LiPo) battery
   - Always check wiring polarity
2. Battery can be charged via USB port or V_{bus}.
3. Battery power is available on LiPo+ or 3.3V pins

L17_03_LoRaGPS

- Notebook: schematic
- Fritzing diagram
- Wire your circuit
- Write the code

Features:

1. Borrow a battery, GPS unit, and LoRa unit from the instructor
2. Using the L14_00_GPS code, obtain GPS coordinates and display on OLED
3. Modify L14_04_LoRaGPS:
   - Integrate the LoRa, GPS, and OLED
   - Get your own RADIOADDRESS from instructors
   - Using IoT_Timer, turn on the D7 LED for 5 seconds when LoRa data received.
   - Display FUSE Sound/Particles to OLED
   - Send live GPS coordinates back to LoRa base-station
   - Find the distance you can go N/S/E/W and get a LoRa signal back to FUSE.
   - Class Trip: repeat at ABQ Biopark

Module 15 - In the Wild

KiCad: Breadboard are good, but PCBs are better

Schematic Capture

KiCad's Schematic Editor supports everything from the most basic schematic to a complex hierarchical design with hundreds of sheets. Create your own custom symbols or use some of the thousands found in the official KiCad library. Verify your design with integrated SPICE simulator and electrical rules checker.

PCB Layout

KiCad's PCB Editor is approachable enough to make your first PCB design easy, and powerful enough for complex modern designs. A powerful interactive router and improved visualization and selection tools make layout tasks easier than ever.

3D Viewer

KiCad's 3D Viewer allows easy inspection of your PCB to check mechanical fit and to preview your finished product. A built-in raytracer with customizable lighting can create realistic images to show off your work.

Start with schematics

Rich library of components

Components
or
Modules

Select a footprint for each component

Generate PCB, Route Wires

Edit board setup including layers, design rules, and various defaults

Update PCB with changes make to schematic

3D View

H2
H3
H1
H4
Red
Yellow
D1
D2
D3
SW1
R1
R2
R3
R4
IoT Bootcamp
PCB Design
Lesson

Import into KiCad

Import Symbols

Using the KiCad (*.lib) file:

1. In KiCad, go to **Tools > Edit Schematic Symbols**.
2. Click on **Preferences > Manage Symbol Libraries**.
3. On the **Global Libraries** tab, click on **Browse Libraries** (the ***small folder icon*** below) and select the .lib file. Then click **Open**. The library will appear, click OK.
4. Toggle the search tree on, and navigate to the symbol you imported. Double-clic over it to open the file.

Import Footprints

Using the *.kicad_*mod* file:

1. In KiCad, go to Tools > Edit PCB Footprints.
2. Click on Preferences > Manage Footprint Libraries.
3. On the **Global Libraries** tab, click on **Browse Libraries** (the ***small folder icon*** below) and navigate to the **Folder** of the downloaded .kicad_mod file. Then click **Open,** and the library will appear. If the path doesn't have the same name. you can rename it as the part.
4. In the table, make sure that the Plugin Type is set to **KiCad**. Then click **OK**.
5. Toggle the search tree on, and navigate to the footprint you imported. Double-click over it to open the file.

Using the **.mod* file:

1. Follow the same steps above from step 1 to step 3.
2. In the table, make sure that the Plugin Type is set to **Legacy**. Then click **OK**.
3. Toggle the search tree on, and navigate to the footprint you imported. Double-click over it to open the file.

Assignment: L15_TrafficPCB

Symbol : Footprint Assignments

1	D1	–	LED	: LED_THT:LED_D3.0mm
2	D2	–	LED	: LED_THT:LED_D3.0mm
3	D3	–	LED	: LED_THT:LED_D3.0mm
4	H1	–	MountingHole	: MountingHole:MountingHole_3.5mm_Pad
5	H2	–	MountingHole	: MountingHole:MountingHole_3.5mm_Pad
6	H3	–	MountingHole	: MountingHole:MountingHole_3.5mm_Pad
7	H4	–	MountingHole	: MountingHole:MountingHole_3.5mm_Pad
8	J1	–	Conn_01x06	: Connector_PinHeader_2.54mm:PinHeader_1x06_P2.54mm_Horizontal
9	R1	–	220	: Resistor_THT:R_Axial_DIN0204_L3.6mm_D1.6mm_P5.08mm_Horizontal
10	R2	–	R	: Resistor_THT:R_Axial_DIN0204_L3.6mm_D1.6mm_P5.08mm_Horizontal
11	R3	–	R	: Resistor_THT:R_Axial_DIN0204_L3.6mm_D1.6mm_P5.08mm_Horizontal
12	R4	–	10k	: Resistor_THT:R_Axial_DIN0204_L3.6mm_D1.6mm_P5.08mm_Horizontal
13	SW1	–	SW_Push	: Button_Switch_THT:SW_PUSH_6mm_H5mm

EN and RST pins

The Argon has two pins that are extremely useful in real world situations:

1. EN pin
   - The EN pin is not a power pin, per se, but it controls the 3V3 power.
   - Device enable pin is internally pulled-up. To disable the device (force the device into a deep power-down state), connect this pin to GND.
   - This pin is essentially an on/off pin.
2. RST pin
   - Active-low system reset input. This pin is internally pulled-up.

Watchdog Timer

The watchdog timer communicates with the MCU at a set interval. If the MCU does not output a signal, outputs too many signals or outputs signals that differ from a predetermined pattern, the timer determines that the MCU is malfunctioning and sends a reset signal to the MCU.

Hardware Watchdog Timers - TPL5010

The timeout frequency is set by resistor R_{EXT} using a formula from the data sheet.

Timeout Interval	Calculated Resistance
1 minute	22 Ω
5 minutes	43 Ω
30 minutes	92 Ω
1 hour	125 Ω
2 hours	170 Ω

Application Watchdog

The Argon also has a watchdog timer that can be implemented in code. It is not as robust as the hardware timer, but better than nothing.

```
// Prototype
// ApplicationWatchdog(unsigned timeout_ms, std::function<void(void)> fn, unsigned
    stack_size=DEFAULT_STACK_SIZE);

// Global variable to hold the watchdog object pointer
ApplicationWatchdog *wd;

void watchdogHandler() {
  // Do as little as possible in this function, preferably just a reset
  System.reset(RESET_NO_WAIT);
}

void setup() {
  // Start watchdog. Reset the system after 60 seconds if the application is unresponsive
  // The stack_size default is 512, but this is too small. Use at least 1536.
  wd = new ApplicationWatchdog(60000, watchdogHandler, 1536);
}

void loop() {
  while (some_long_process_within_loop) {
    ApplicationWatchdog::checkin(); // resets the AWDT count
  }
}
// AWDT count reset automatically after loop() ends
```

Solar Charging

Should be easy, but isn't.

Power Management

System.sleep() can be used to lower power consumption and increase battery life when the Particle is only needed intermittently to interact with the environment.

	STOP	ULTRA_LOW_POWER	HIBERNATE
Relative power consumption	Low	Lower	Lowest
Relative wake options	Most	Some	Fewest
Execution continues with variables intact	✓	✓	

Wake Mode	Gen 2	Gen 3
GPIO	✓	✓
Time (RTC)	✓	✓
Analog		✓
Serial		✓
BLE		✓
Cellular		✓
Wi-Fi		✓

Ultra Low Power example

The below code places the Argon in Ultra_Low_Power mode, enabling it to be awakened either by a RISING signal on Pin D0 or after the time specified by the variable sleepDuration.

```
void sleepULP() {
  SystemSleepConfiguration config;
  config.mode(SystemSleepMode::ULTRA_LOW_POWER).gpio(D0,RISING).duration(sleepDuration);
  SystemSleepResult result = System.sleep(config);
  delay(1000);
  if (result.wakeupReason() == SystemSleepWakeupReason::BY_GPIO) {
    Serial.printf("Awakened by GPIO %i\n",result.wakeupPin());
  }
  if (result.wakeupReason() == SystemSleepWakeupReason::BY_RTC) {
    Serial.printf("Awakened by RTC\n");
  }
}

/*
 * Other wake-up modes
 *
 *  config.mode(SystemSleepMode::ULTRA_LOW_POWER).analog(pin, value, mode);
 *    mode can be: ABOVE, BELOW, or CROSS
 *  config.mode(SystemSleepMode::ULTRA_LOW_POWER).usart(Serial1);
 *    note: Serial can not be used for wake-up
 *  config.mode(SystemSleepMode::ULTRA_LOW_POWER).ble();
 *  config.mode(SystemSleepMode::ULTRA_LOW_POWER).network(NETWORK_INTERFACE_WIFI_STA);
 */
```

Cloud Flash

During the deployment phase, it isn't convenient to have to hook up a USB cable to push updates. The Particle ecosystem allows for sending code over-the-air (OTA).

- Cloud Compile - compile your program and download the binary
- Cloud Flash - compile and flash it to the selected device OTA

The OTA operations require:

- The device is connected to the Particle Cloud (breathing cyan)
- The computer is into the same account that claimed the device.
- The DeviceID is set to the device name.

Local MQTT Server

mosquitto

Node-RED

Install Mosquitto (Basic)

```
// On Windows go to mosquitto.org/download

// On Mac using Homebrew
brew install mosquitto

// On Linux (Debian and Ubuntu)
sudo apt-get install mosquitto mosquitto-clients

// test broker
// open new terminal window
mosquitto_sub -h localhost -t feeds/test
// in the original terminal
mosquitto_pub -h localhost -t feeds/test -m "Hello
    World"
```

Install Node-Red (Basic)

```
// Start With Installing nodejs:

// On Windows go to nodejs.org/en/

// On Mac using Homebrew
brew install node

// On Linux (Debian and Ubuntu)
sudo apt install nodejs npm

// VERFIY: In terminal or PowerShell, verify using
node -v
npm -v

// Install Node-RED (all platforms)
npm install -g --unsafe-perm node-red
```

In a browser URL bar, go to localhost:1880

Node Red Dashbaard

Node Red MQTT Node

Node Red MQTT Connection

Edit mqtt in node > **Edit mqtt-broker node**

Delete | Cancel | Update

Properties

Name: ddciot.us

Connection | Security | Messages

Server: mqtt.ddciot.us Port: 1883

☑ Connect automatically

☐ Use TLS

Protocol: MQTT V3.1.1

Client ID: Leave blank for auto generated

Keep Alive: 60

Session: ☑ Use clean session

Edit mqtt in node > **Edit mqtt-broker node**

Delete | Cancel | Update

Properties

Name: ddciot.us

Connection | **Security** | Messages

Username: fuse

Password: ••••••••

Node Red Smart Fuse Flow

Node Red Smart Fuse Dashboard

Install Mosquitto MQTT Broker (Detailed: Linux)

Install Mosquitto

```
// install Mosquitto broker and client
sudo apt-get install mosquitto mosquitto-clients
// test broker
// open new terminal window
mosquitto_sub -h localhost -t feeds/test
// in the original terminal
mosquitto_pub -h localhost -t feeds/test -m "Hello World"
// create a password file - replace bob with your username
// the -c overwrites existing file, leave off to append
sudo mosquitto_passwd -c /etc/mosquitto/passwd bob
```

Create and edit file: /etc/mosquitto/conf.d/default.conf

```
// File: /etc/mosquitto/conf.d/default.conf
allow_anonymous false
password_file /etc/mosquitto/passwd
```

Test the username and password

```
// restart mosquitto
sudo systemctl restart mosquitto
// this should create an error now
mosquitto_pub -h localhost -t feeds/test -m "Hello World"
// instead exit sub and restart both sub and pub using username and password above
mosquitto_sub -h localhost -t feeds/test -u "bob" -P "password"
mosquitto_pub -h localhost -t feeds/test -m "Hello World" -u "bob" -P "password"
```

Node-RED (Detailed: Linux)

```
// Install nodejs and npm
sudo apt install nodejs npm
node -v // validate it was installed correctly

// Install Node-RED
sudo npm install -g --unsafe-perm node-red

// Start node-red service on reboot using PM2
sudo npm install -g pm2

// find and use the correct node-red path
which node-red
pm2 start /usr/local/bin/node-red -- -v
pm2 save
pm2 startup // and follow instructions

// some systems might require
pm2 startup systemd
```

Securing Node-Red (Detailed: Linux)

```
// Install the node-red-admin package
sudo npm install -g --unsafe-perm node-red-admin
// Create a password hash
node-red-admin hash-pw
```

Edit /.node-red/settings.js

```
// File: ~/.node-red/settings.js
// Find and uncommnet the adminAuth section using the hash
  adminAuth: {
    type: "credentials",
    users: [{
      username: "admin",
      password: "$2a$08$zZWtXTja0fB1pzD4sHCMyOCMYz2Z6dNbM6tl8sJogENOMcxWV9DN.",
      permissions: "*"
    },
    {
      username: "bob",
      password: "$2b$08$W7tKOfDNkm6eZUucgFMoauV1qPwf60MnGzumCc/B8Xj/FUjs8SVMq",
      permissions: "*"
    }]
  },
```

Restart node-red to allow all changes to take effect

```
sudo systemctl restart node-red
```

Accessing from other computers (on same network)

```
// Find your systems IP Address
sudo apt install net-tools
ifconfig

// Open ports from Mosquitto and Node-Red
sudo ufw allow 1880 // Node-Red port
sudo ufw allow 1883 // MQTT port
sudo ufw enable     // Turn on firewall
sudo ufw status
```

- Devices and publish/subscribe to your Mosquitto MQTT Broker if on the same network using your computer's IP address and Port: 1883.
- Access Node-Red from browser on same network using IPAddress:1880

Note: To access anywhere, you need to setup on a cloud service such as DigitalOcean.

Capstone

Capstone Projects

Intent:

- Based on a direct observation or need expressed by a guest speakers.
- Original work demonstrating the skills obtained in this class.
- Demonstrate the ability to work as part of a team.
- A pitch to potential employers or investors.

Guidelines:

- Practical application of smart home, manufacturing, community environment, or immersive entertainment.
- Code MUST follow the IoT Style Guide
- Needs to include a Cloud Dashboard component and concepts from Module 15 - In the Wild
- Project will include a video, presentation, GitHub, and Hackster.io.

Previous capstone projects can been found at: `https://www.youtube.com/playlist?list=PL0t2Pk5ETDgxfVptdyr6xbL6MW1-5CJey`

Extra

Particle Libraries

Creating and Publishing Your Own Libraries

```
// Within your project directory, create a library
mkdir mylib
cd mylib
particle library create

// Modify the Project.Properties file, especially version number

// Create your .h, .cpp, and/or examples within the library directory structure

// Upload your library
particle library upload

// Publish your library
particle library publish mylib

// Note: you can upload/publish new versions
//       just change the version number in Project.Properties
```

Non-Embedded C++

HelloWorld in C++

```
// include the standar input-output library
// most include statements need the .h, but iostream is an exception
#include <iostream>

// namespace is used to declare regions with the global space
// std enable the standard console (monitor) and input (keyboard)
using namespace std;

char myName[20];

//main is the first function executed in your cpp code
int main()
{
    cout << "Hello World!!!\n";    // cout = output to console
    cout << "What is your name: ";
    cin >> myName;                 // cin = input from console
    cout << "Hello " << myName << ", I hope you are having a nice day.\n";

    return 0;  // denotes successfully executed
}
```

Instructions for installing and writing C++ code in VSCode:
https://code.visualstudio.com/docs/languages/cpp

The Big Question: What about main()

```
#include <Arduino.h>

extern "C" int main(void)
{
#ifdef USING_MAKEFILE

  // To use Teensy 3.0 without Arduino, simply put your code here.
  // For example:

  pinMode(13, OUTPUT);
  while (1) {
    digitalWriteFast(13, HIGH);
    delay(500);
    digitalWriteFast(13, LOW);
    delay(500);
  }


#else
  // Arduino's main() function just calls setup() and loop()....
  setup();
  while (1) {
    loop();
    yield();
  }
#endif
}
```

Hello World - What a microcontroller sees

HelloWorld.ino

```
void setup() {
  Serial.begin(9600);
  Serial.printf("Hello World! \n");
}

void loop() {}
```

Hex Code and Assembly Language

```
HelloWorld.bin:     file format binary
Disassembly of section .data:
00000000 <.data>:
    101c: bd10        pop {r4, pc}
    101e: 4402        add r2, r0
    1020: 4603        mov r3, r0
    1022: 4293        cmp r3, r2
    1024: d002        beq.n 0x102c
    1026: f803 1b01   strb.w  r1, [r3], #1
    102a: e7fa        b.n 0x1022
    102c: 4770        bx  lr
    102e: 0000        movs  r0, r0
    1030: b538        push  {r3, r4, r5, lr}
```

Useful BASH commands

Redirect from stdout (standard output)

The > and >> signs are used for redirecting the output of a program to something other than stdout (standard output, which is the terminal by default).

- The >> appends to a file or creates the file if it doesn't exist.
- The > overwrites the file if it exists or creates it if it doesn't exist.

Examples:

```

# Create a file called "allmyfiles.txt" and fill with the directory listing

ls > allmyfiles.txt

# Adds "End the directory listing" to the end of "allmyfiles.txt"

echo "End of directory listing" >> allmyfiles.txt

# Create a zero-byte file with the name "newfile.txt"

> newfile.txt

# Redirect Particle Serial Monitor output to the file "filename.csv"

Particle serial monitor --follow >> filename.csv
```